A Fast Solver-Free Algorithm for Traffic Engineering in Large-Scale Data Center Network

Yingming Mao[1,2], Qiaozhu Zhai[1], Ximeng Liu[3], Zhen Yao[4], Xia Zhu[4], Yuzhou Zhou[1]
[1]Xi'an Jiaotong University [2]Shanghai Innovation Institute [3]Shanghai Jiao Tong University [4]Huawei

Abstract

Rapid growth of data center networks (DCNs) poses significant challenges for large-scale traffic engineering (TE). Existing acceleration strategies, which rely on commercial solvers or deep learning, face scalability issues and struggle with degrading performance or long computational time.

Unlike existing algorithms adopting parallel strategies, we propose Sequential Source-Destination Optimization (SSDO), a sequential solver-free algorithm for intra-DCN TE. SSDO decomposes the problem into subproblems, each focused on adjusting the split ratios for a specific source-destination (SD) demand while keeping others fixed. To enhance the efficiency of subproblem optimization, we design a Balanced Binary Search Method (BBSM), which identifies the most balanced split ratios among multiple solutions that minimize Maximum Link Utilization (MLU). SSDO dynamically updates the sequence of SDs based on real-time utilization, which accelerates convergence and enhances solution quality.

We evaluate SSDO primarily on Meta DCNs, and additionally on two WAN topologies as auxiliary demonstrations of generality. In a Meta topology, SSDO achieves a 65% and 60% reduction in normalized MLU compared to TEAL and POP, two state-of-the-art TE acceleration methods, while delivering a 12× speedup over POP. These results demonstrate the superior performance of SSDO in large-scale TE.

1 Introduction

With the rapid development of social networks and large language models (LLMs) [37], data center networks (DCNs) face increasingly demanding performance requirements. To address this, companies like Microsoft [19] and Google [36] have adopted centralized Traffic Engineering (TE) systems powered by Software-Defined Networking (SDN) [2, 3, 10, 21, 26, 29, 45]. These systems optimize traffic routing across fixed network paths to improve performance, often formulating TE as multicommodity flow problems [31] to minimize Maximum Link Utilization (MLU) or maximize network flow, solved by a centralized controller [4, 5, 14].

The TE controller operates by collecting traffic demands and solving a linear programming [31] to determine traffic allocations. This periodic process ensures that the routing of traffic aligns with real-time demands [36, 48]. However, as DCNs scale to hundreds of nodes and tens of thousands of edges, the computational overhead grows significantly, making real-time TE increasingly challenging.

Contemporary traffic engineering (TE) acceleration methods can be broadly categorized into two approaches: linear programming (LP)-based and deep learning (DL)-based methods. LP-based algorithms accelerate TE by decomposing the TE optimization problem into smaller subproblems based on demands or topologies [1, 33], which are solved concurrently. However, this often results in degrading TE quality, as neglecting the coupling between subproblems. DL-based methods, such as Teal [44] leverage historical data to directly map traffic matrices to TE configurations, significantly accelerating the computation process. However, these methods face challenges such as dependence on the quality and diversity of training data and may struggle to generalize to unseen traffic patterns or network conditions.

In contrast to conventional acceleration algorithms, our key insight is to address the coupling between subproblems by solving them in a carefully designed sequence, where each subproblem builds on the solution of the previous one. Unlike parallel schemes, which often struggle to maintain global coherence, the sequential strategy progressively incorporates global network information by following a structured optimization order. This iterative refinement stabilizes at a high-quality solution while mitigating the degradation issues that commonly hinder parallel methods, making it a more reliable alternative.

Sequential TE algorithms require each subproblem to be solved efficiently, as cumulative computation time can become a bottleneck. Thus, Sequential Source-Destination Optimization (SSDO) was proposed, which decomposes the original problem into subproblems, each optimizing the split ratios for a specific source-destination (SD). This structure enables the design of a binary search-based algorithm, avoiding the

high complexity of LP solvers. However, subproblems often have multiple valid solutions, and selecting an unsuitable one can slow convergence and degrade TE quality, making LP solvers unsuitable for subproblem solving. To mitigate this, we develop the Balanced Binary Search Method (BBSM), which not only accelerates subproblem solving but also ensures that selected solutions enhance subsequent optimization.

In addition, SSDO adopts a dynamic optimization sequence that prioritizes edges with the highest utilization. In each iteration, it identifies the most congested edges and selects all SDs whose paths traverse them. The corresponding subproblems are then solved to adjust split ratios, reducing congestion. After each step, edge utilization is updated to guide subsequent optimizations, ensuring that SSDO continuously focuses on the most constrained parts of the network and accelerates convergence toward higher-quality solutions. Moreover, since SSDO ensures a non-increasing MLU during optimization, terminating the algorithm at any point guarantees a solution that is at least improved compared to the initial configuration.

We evaluate SSDO with Meta DCNs and various wide-area network (WAN) topologies. SSDO offers a better balance of computation time and TE quality than existing algorithms. On the Meta-Web topology with a per-pair four-path limit, SSDO reduces solution time by 92% relative to LP, with an error of less than 1%. It also reduces error by 60% and time by 90% against POP [33], a state-of-the-art LP-based acceleration method. In topologies that are too large for DL-based methods, SSDO consistently delivers efficient and high-quality solutions. Our code is available at [41].

This work does not raise any ethical issues.

2 Background and Motivation

2.1 Existing Methods Facing Scale Challenge

The rapid expansion of networks has made large-scale TE increasingly challenging. As an LP problem, allocating traffic across paths containing hundreds of nodes often requires several hours using commercial solvers. Consequently, operators are seeking methods to accelerate TE optimization.

LP-based direct methods. Traditionally, TE is modeled as a multicommodity flow problem [31] and solved using commercial LP solvers due to its modest scale in earlier networks. However, with the expansion of data center networks, the computational overhead of LP solvers has become prohibitive. The worst-case complexity of LP is regarded as $O(n^3)$ ($O(n^{2.373})$ in [28, 33]), making it impractical for large-scale networks. For example, in a fully connected network with 150 nodes, assuming four paths per SD, LP requires solving for $4 \times 150 \times 149 = 89,400$ variables. This leads to substantial memory usage and long computational times. Commercial solvers attempt to accelerate computations by launching multiple threads, each running a different optimization algorithm independently. The solver then selects the solution from the fastest-converging algorithm. However, their acceleration relies on executing multiple optimization methods in parallel and selecting only the fastest one, which inherently limits performance improvements.

DL-based direct methods. DL approaches, such as DOTE [35] and Figret [30], have been introduced to accelerate TE using MLU as the loss function. Although these methods demonstrate efficiency in limited-scale DCNs, their performance deteriorates significantly at larger scales. For example, in the same scenario of 89,400 variables, the DL model must output all variables in the output layer, which greatly hampers its generalization due to the "curse of dimensionality" [25]. This constraint makes DL-based direct methods ill-suited for scaling up to large network sizes.

LP-Based parallel accelerating methods. Parallel methods have emerged as promising solutions to accelerate TE processes. For example, the POP method [33] decomposes the optimization problem into k subproblems, each preserving the network topology but handling only a subset of demands. Similarly, NCFlow [1] partitions both the demands and the network topology into k distinct clusters. These methods solve all subproblems simultaneously by invoking LP solvers and then combine their solutions to approximate an acceptable feasible solution. Increasing k can significantly reduce computational time, but this comes at the cost of degrading TE performance due to the coupling between subproblems. This trade-off between computation time and solution quality is a critical limitation of parallel LP-based approaches.

DL-based parallel accelerating methods. To alleviate the "curse of dimensionality" in DL methods, Teal [44] was introduced. Similar to POP, Teal utilizes a shared policy network to independently compute split ratios for each demand. Additionally, Teal incorporates a multi-agent reinforcement learning (MARL) strategy to manage coupling among demands. Despite its advancements, the efficacy of Teal is significantly dependent on the correlation between historical and future traffic matrices and the generalizability of the shared policy network. These factors may result in degradation within complex network environments.

2.2 Accelerate TE with Sequential Strategy

Due to the difficulty of parallel strategies in addressing the coupling between subproblems, we propose a sequential strategy to optimize traffic allocation. By decomposing the problem into subproblems, each modifying the split ratios for a specific SD, and determining an appropriate solving order, the sequential strategy has the potential to achieve higher-quality traffic allocations compared to parallel strategies.

Better handling of subproblem coupling. Unlike parallel methods that solve subproblems simultaneously but struggle with global coherence, our approach addresses subproblems sequentially, with each decision based on the previous one. This allows each subproblem to progressively capture the

overall state of the network. By structuring the solving order, the sequential approach better accounts for subproblem coupling, leading to better performance than parallel methods.

Direct inheritance of existing algorithm Results. Due to the monotonic nature of the proposed sequential algorithm, when initialized with a TE configuration derived from other methods, the resulting performance will always be at least as good as the original configuration. This ensures compatibility with previous approaches while enabling further improvement.

Leveraging all available computing time. The adjustment cycles for split ratios vary significantly across different networks, ranging from 10 seconds to 15 minutes, posing challenges for TE. LP-based parallel approaches require selecting k, the number of subproblems, to fit within the given cycle. However, a smaller k improves precision but increases complexity, potentially exceeding the adjustment cycle, while a larger k simplifies subproblems but sacrifices precision, degrading solution quality. Similarly, DL-based methods, while fast, inherently lack mechanisms to utilize unused computing time for further refinement. Once the solution is computed, any remaining adjustment time is left idle. In contrast, SSDO adapts seamlessly to varying adjustment cycles by performing high-frequency updates to split ratios starting from an initial feasible TE configuration. This approach ensures consistent improvement for short cycles while fully utilizing longer cycles for further refinement, enabling superior configurations under different computation time constraints.

2.3 Key Challenges in Designing Effective Sequential Strategies

While sequential strategies have the potential to achieve high-quality solutions, their implementation presents significant challenges. Designing an effective sequential approach requires addressing key issues related to computation time, solution consistency, and task sequencing.

Computing efficiency for subproblems. Sequential strategies solve subproblems one by one, making efficiency critical, especially when the number of subproblems is large. Although commercial solvers such as CPLEX and Gurobi offer efficient methods for solving optimization problems, their overhead in model construction and complex solving processes make them impractical for handling individual subproblems in a sequential framework.

Inconsistency between subproblem and global performance. Decisions made in early subproblems can constrain the solution space for later ones, potentially leading to poor global performance. This lack of coordination often results in inferior overall outcomes, requiring additional adjustments to improve global performance.

Impact of subproblem order. The sequence in which subproblems are solved significantly affects convergence speed and solution quality. While a random order can yield improvements over initial conditions, an inefficient order may slow convergence, requiring more iterations to achieve satisfactory results. Identifying an effective order is critical for improving solution quality and computational efficiency.

3 TE MODEL

Notations & Definitions: We present the recurrent mathematical notations and definitions pertaining to TE. As this work focuses on TE in DCNs, we focus on one-hop and two-hop transit paths, which suffice for most DCN deployments [36, 48]. The generalization to multi-hop scenarios, typical of WANs, is given in Appendix A.

- **Network.** The network topology is a graph $G = (V, E, c)$, with V as vertices, E as edges and c_{ij} specifying the sum of capacities from vertices i to j.

- **Traffic demands.** The Demand matrix, denoted as D, is a $|V| \times |V|$ matrix where each element D_{ij} represents the traffic demand from source i to destination j.

- **TE configuration.** TE configuration $\mathcal{R}$ outlines the split ratio, indicated as f_{ikj}, which expresses the proportion of traffic from the source i to the destination j that crosses an intermediary node k. This 3D matrix compactly encodes split-ratio information, representing the distribution of traffic across routing paths. Formally:

  - f_{ikj}: Represents the fraction of traffic from i to j that follows a two-hop path through k, where $i \neq k \neq j$.
  - f_{ijj}: Denotes the fraction of traffic directly routed from i to j (1-hop path), where $i \neq j$.
  - f_{iij} and f_{iki}: Since the direct path is already captured by f_{ijj}, and self-traffic is not considered, $f_{iij} = f_{iki} = 0$.

  This 3D matrix stores split ratio information densely, providing a strong basis for future calculations.

- **Path set.** Practical TE systems typically constrain the set of paths available between SDs due to network topology or operational policies. The path set, denoted as $\mathcal{P}$, represents all permissible routing paths. Each element of $\mathcal{P}$ is an ordered triad of nodes, such as (s, k, d), representing a valid path between source s and destination d via intermediate node k. If traffic follows a direct path, we set $k = d$. For a given (s, d), we define $\mathcal{K}_{sd}$ as the set of intermediate nodes k associated with the paths in $\mathcal{P}$. Specifically, $\mathcal{K}_{sd} = \{k \mid (s, k, d) \in \mathcal{P}\}$.

- **TE objective.** The objective function explored in this study aims to minimize MLU, denoted u, a metric widely used in TE [8, 9, 12, 36, 42]. It effectively encapsulates both throughput and resilience to traffic fluctuations. MLU is defined as $\max_{i,j \in V} \left(\sum_{k \in V} f_{ijk} \cdot D_{ik} + \sum_{k \in V} f_{kij} \cdot D_{kj}\right)/c_{ij}$, which is calculated by the given Demand matrix D and the TE configuration $\mathcal{R}$.

Optimization model of TE: The TE problem can be formulated as a linear programming (LP) problem, where the goal is to determine the optimal split ratios to minimize MLU while satisfying flow conservation constraints. The optimization model is defined as Equation (1).

$$\min_{f_{ikj}\in\mathcal{R}} \; u$$
$$\text{s.t.} \quad \begin{cases} f_{ikj} \geq 0, \quad f_{iki} = 0, \quad f_{iij} = 0, & \forall i,j,k \in V, \\ f_{ikj} = 0, & \forall (i,k,j) \notin \mathcal{P}, \\ \sum_{k\in V} f_{ikj} = 1, & \forall i \neq j \in V, \\ \frac{\sum_{k\in V} f_{ijk}\cdot D_{ik} + \sum_{k\in V} f_{kij}\cdot D_{kj}}{c_{ij}} \leq u, & \forall i \neq j \in V. \end{cases} \quad (1)$$

4 SSDO Design

4.1 Overview

As illustrated in Figure 1, SSDO takes predetermined split ratios and traffic demand as input. The *SD Selection* component identifies SDs based on the current split ratios and traffic demands. The *Split Ratio Modification* component then optimizes the split ratios for the selected SD. This iterative process alternates between *SD Selection* and *Split Ratio Modification*. With each iteration, the system's MLU progressively decreases, ultimately converging to a high-quality solution.

For a given SD (s,d), the *Split Ratio Modification* component formulates a subproblem with $f_{skd}, \forall k \in \mathcal{K}_{sd}$ as decision variables, while keeping other split ratios fixed. Instead of solving it as an LP problem, SSDO reformulates it as a structured search problem, significantly reducing computational complexity. LP solvers rely on costly matrix operations and iterative constraint satisfaction, often requiring $O(n^3)$ complexity for large-scale problems. In contrast, SSDO employs a binary search algorithm, which converges in logarithmic time with only a few function evaluations, avoiding the high overhead of traditional optimization techniques. To ensure subproblem solutions align with global TE performance, SSDO selects the most balanced solution among the subproblem's optima. A detailed description is provided in §4.2.

The *SD Selection* component in SSDO identifies the set of edges with maximal utilization, determined by the split ratios and demands. It then locates the associated SDs and provides them to the *Split Ratio Modification* component. Without a well-designed selection procedure, the process could converge slowly or settle into inferior local optima. SSDO's carefully crafted rules largely avert these pitfalls, as we detail in §4.3.

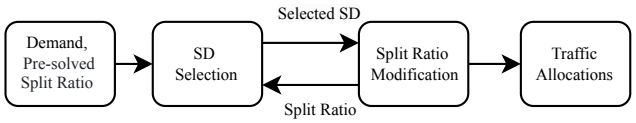


Figure 1: Workflow of SSDO.

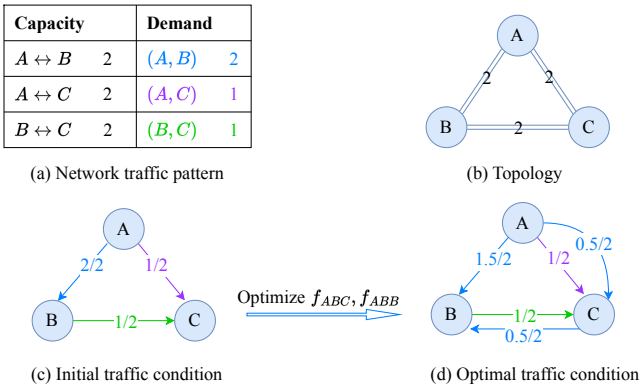


Figure 2: A sample illustration of the subproblem optimization (SO). Notations like "1/2" on the edges mean that the flow through the edge is 1 and the capacity of the edge is 2. In this example, only one SO is required for the SSDO algorithm. In initial TE scheme, $f_{ABB} = 100\%$, $f_{ACB} = 0\%$, $f_{ACC} = 100\%$, $f_{ABC} = 0\%$, $f_{BCC} = 100\%$, $f_{BAC} = 0\%$. After SO process, f_{ABB} change to 75%, f_{ACB} change to 25%.

In addition, SSDO can be initiated with any feasible presolved split ratios. A potential approach to constructing this solution is to route each SD's demand entirely along one of its available paths. All components of SSDO are meticulously designed, necessitating only basic matrix operations of addition and multiplication. SSDO does not require historical data or significant computational resources, making it straightforward to program and implement.

4.2 Split Ratio Modification Component

Subproblem definition. In this section, we focus on an LP subproblem of TE. In the subproblem, only the split ratios related to the selected SD are subjected to optimization, while all other split ratios remain constant, which is called subproblem optimization (SO). To elucidate the fundamental concept of SO, the process is illustrated in Figure 2. Within this network, there are three SDs: (A,B), (B,C), and (A,C). The initial TE scheme routes all traffic along the shortest paths, resulting in an MLU of $\max\{1, 0.5, 0.5\} = 1$, which occurs at the edge $A \to B$. By altering the split ratios for (A,B) and maintaining those for (B,C) and (A,C) unchanged, the MLU transitions to $\max\{0.75, 0.75, 0.5, 0.25\} = 0.75$. In particular, 0.75 represents the minimum MLU achievable in this system under the given traffic pattern.

Subproblem characteristics. Compared to the original problem like Equation (1), the SO problem of given SD (s,d) requires optimizing only the $|\mathcal{K}_{sd}|$ split ratios, significantly simplifying the problem. From a programming perspective, the SO problem remains an LP problem. Fortunately, it has some unique characteristics that can be further leveraged to simplify the calculation.

Characteristic 1: ***Without solving SO, the feasibility of a given MLU u_0 can be analytically judged.***

The feasibility of a given MLU u_0 can be judged without solving SO. This process is illustrated in Figure 3 and involves the following steps:

1. **Background traffic computation:** Suppose that the selected SD is designated as (s,d) . Background traffic from node i to node j, denoted as Q_{ij}, can be determined by setting $f_{skd} = 0$ for all $k \in V$, as in Equation (2). The calculation example is shown in (b) of Figure 3.

$$Q_{ij} = \begin{cases} \sum_{k\in V} f_{ijk}\cdot D_{ik} + \sum_{k\in V} f_{kij}\cdot D_{kj}, & i \neq s, j \neq d \\ \sum_{k\in V/d} f_{ijk}\cdot D_{ik} + \sum_{k\in V/s} f_{kij}\cdot D_{kj}, & i = s, j = d \\ \sum_{k\in V/d} f_{ijk}\cdot D_{ik} + \sum_{k\in V} f_{kij}\cdot D_{kj}, & i = s, j \neq d \\ \sum_{k\in V} f_{ijk}\cdot D_{ik} + \sum_{k\in V/s} f_{kij}\cdot D_{kj}, & i \neq s, j = d \end{cases} \tag{2}$$

2. **Residual capacity calculation:** For a given path $s \to k \to d$, the residual capacity T_{skd} is computed using Equation (3). Here, T_{skd} represents the maximum remaining capacity of the path, calculated by the background traffic Q and the given u_0. Based on this residual capacity, the upper bound of the split ratio through k, denoted as $\bar{f}_{skd}$, is derived using Equation (4).

$$T_{skd} = \begin{cases} \min\left\{ \begin{array}{l} u_0 c_{sk} - Q_{sk}, \\ u_0 c_{kd} - Q_{kd} \end{array} \right\}, & k \in \mathcal{K}_{sd}, k \neq d, \\ u_0 c_{sd} - Q_{sd}, & k = d \end{cases} \tag{3}$$

$$\bar{f}_{skd} = \frac{T_{skd}}{D_{sd}}, \tag{4}$$

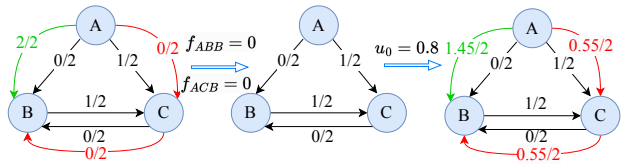
3. **Feasibility assessment:** Drawing from the preceding analysis, the feasibility of SO can be evaluated through the following metrics.

   - If $\sum_{k\in\mathcal{K}_{sd}} \bar{f}_{skd} \geq 1$ and $\min_{k\in\mathcal{K}_{sd}} \bar{f}_{skd} \geq 0$, there is a feasible solution. In this case, $\bar{f}_{skd}$ can be normalized to determine the solution, as shown in Figure 3.
   - If $\sum_{k\in\mathcal{K}_{sd}} \bar{f}_{skd} < 1$ or $\min_{k\in\mathcal{K}_{sd}} \bar{f}_{skd} < 0$, the given u_0 lies outside the feasible domain.

Characteristic 2: ***The optimal MLU u^* in SO can be determined by binary search.***

Based on the analysis above, the upper bound of the split ratio $\bar{f}_{skd}$ is fundamentally related to the MLU parameter u. As rigorously proven in Appendix D, each individual $\bar{f}_{skd}(u)$ is a nondecreasing function of u. This component-wise monotonicity implies that for any intermediate node $k \in \mathcal{K}_{sd}$, we have:

$$\bar{f}_{skd}(u) \geq \bar{f}_{skd}(u_0) \quad \text{whenever } u \geq u_0. \tag{5}$$



(a) Initial traffic condition (b) Background traffic (c) Feasible traffic condition

Figure 3: Illustration of judgment process of SO proposed in Figure 2 when $u_0 = 0.8$, $D_{AB} = 2$. The green and red lines represent the traffic flows on $A \to B$ and $A \to C \to B$. Let $f_{ABB} = f_{ACB} = 0$, background traffic Q is calculated in (b). Using background traffic, $T_{ACB} = \min\{2\times 0.8 - 1, 2\times 0.8 - 0\} = 0.6$, $T_{ABB} = 0.8 \times 2 = 1.6$, $\bar{f}_{ACB} = 0.6/2 = 0.3$, $\bar{f}_{ABB} = 1.6/2 = 0.8$. To satisfy the normalization constraint, $f_{ACB}, f_{ABB} = 0.3/(0.8+0.3), 0.8/(0.8+0.3)$, a feasible solution having been obtained in (c).

The aggregation of these monotonic components preserves the nondecreasing property. Specifically, summing over all possible paths $k \in \mathcal{K}_{sd}$ yields:

$$\sum_{k\in\mathcal{K}_{sd}} \bar{f}_{skd}(u) \geq \sum_{k\in\mathcal{K}_{sd}} \bar{f}_{skd}(u_0) \quad \text{whenever } u \geq u_0. \tag{6}$$

This monotonicity ensures that if u_0 is feasible, then all $u \geq u_0$ are also feasible. Conversely, if u_0 is infeasible, then all $u \leq u_0$ are also infeasible.

To perform a binary search, we must define the lower and upper boundaries u^{lb} and u^{ub}, which ensure a bounded search space. These boundaries are given as Equation (7) and Equation (8). u^{lb} represents the minimum possible MLU, below which the solution becomes infeasible. Specifically, for $u < u^{lb}$, the split ratio $\bar{f}_{skd}$ would become negative, violating the feasibility conditions. u^{ub} provides the maximum feasible MLU under initial conditions before modification. This ensures that any feasible solution must lie within $[u^{lb}, u^{ub}]$.

$$u^{lb} = \max_{i,j\in V} \frac{Q_{ij}}{c_{ij}}, \tag{7}$$

$$u^{ub} = \max_{i,j\in V} \frac{\sum_{k\in V} f_{kij}\cdot D_{kj} + \sum_{k\in V} f_{ijk}\cdot D_{ik}}{c_{ij}}. \tag{8}$$

With these boundaries established, we can conclude that there exists a threshold $u^* \in [u^{lb}, u^{ub}]$ such that u^* is the optimal MLU. The monotonicity of $\bar{f}_{ikj}(u)$ further guarantees the correctness of the binary search within this range. Thus, the above analysis ensures that the binary search can determine not only feasible but also optimal MLU u^* in SO.

Characteristic 3: ***For the optimal MLU u^*, there exist multiple feasible TE configurations, but only one balanced TE configuration which can be binary searched.***

As illustrated in Figure 4, the multi-solution phenomenon for split ratios occurs if and only if the optimal MLU u^* equals the lower bound u^{lb}. Otherwise ($u^* > u^{lb}$), the solution is unique. Under this specific condition, multiple sets of

split ratios can achieve the same u^*, resulting in ambiguity in the solution. To address this issue and better coordinate the performance of SO and origin optimization, we introduce 'balance' as a secondary objective in SO. The balanced solution is formulated to satisfy the following two key conditions.

- For each path with non-zero split ratios, the maximum utilization of its edges equals a fixed value u^e.
- For each path with zero split ratios, the maximum utilization of its edges exceeds or equals u^e .

An example of this balanced solution is shown in (c) of Figure 4.For (A,B), f_{ACB} and f_{ADB} are greater than zero, the maximum utilization of the paths $A \to C \to B$ and $A \to D \to B$ is equal to 0.55, satisfying the first condition. Furthermore, the maximum utilization of $A \to B$ exceeds 0.55, fulfilling the second condition. In contrast, an alternative solution shown in (d) of Figure 4 fails to meet the second condition, as the maximum utilization of the paths $A \to D \to B$ does not exceed the threshold u^e, highlighting its imbalance in this scenario. By ensuring that the balanced solution meets these conditions, it not only resolves the ambiguity caused by the multisolution phenomenon, but also guarantees a more balanced distribution of traffic across paths.

The introduction of u^e provides significant benefits in the optimization process.

- **Providing more optimization potential.** Without u^e, the SO process cannot effectively determine which solution among multiple feasible configurations is optimal for the overall TE objective. Blindly increasing the traffic on certain edges may severely restrict the optimization space for subsequent SDs, leading the algorithm to converge on inferior solutions. By balancing capacity utilization across edges, u^e helps avoid such pitfalls. Although it may result in a time cost for finding u^e, this balanced approach ensures that the solution space remains flexible, preventing the algorithm from being trapped in poor feasible configurations.
- **Seamless integration into the binary search framework.** Like the optimal MLU u^*, the balanced MLU u^e can also be determined via binary search. The key difference is that u^e is based on the modified upper bounds:

$$\bar{f}^b_{skd}(u) = \max\{0, \bar{f}_{skd}(u)\}, \tag{9}$$

  which exclude negative split ratios. The search space of u^e is bounded between 0 and u^{ub} (the same upper bound as u^*). In single-solution cases, u^e coincides with u^*; in multi-solution cases, binary search on u^e directly yields the balanced solution. This formulation transforms SO into a binary search problem while ensuring both computational efficiency and robustness.

Balanced binary search method for SO. To efficiently solve the SO problem, we propose a balanced binary search algorithm (BBSM), as detailed in Algorithm 1. The algorithm is

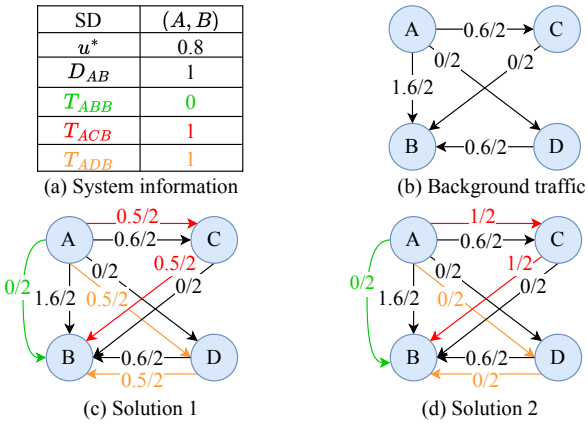


Figure 4: An illustration of the multi-solution phenomenon of SO. In this SO, using multiple split ratios will obtain the same MLU.

designed to leverage the characteristics of the problem for improved computational efficiency. Specifically, apart from the initialization step, all operations within BBSM have a time complexity of $O(|V|)$. The binary search process is controlled by a threshold ε, typically set to 10^{-6}, ensuring convergence within approximately $\log_2(1/\varepsilon) = 20$ iterations.

For the initialization phase, if the method in Equation (2) is applied to calculate Q, the time complexity reaches $O(|V|^3)$. However, in practice, this complexity can be reduced to $O(|V|)$ by maintaining a utilization matrix and updating the corresponding path utilization dynamically based on the selected SD. This practical implementation significantly reduces computational time overhead.

In contrast to the linear programming approach, whose computational complexity scales as $O(n^3) \approx O(|V|^9)$ in a fully connected network, and which does not explicitly prioritize among multiple equally optimal solutions, the proposed BBSM demonstrates superior performance. With its lower computational complexity and its ability to identify well-balanced solutions among multiple feasible options, BBSM provides a more efficient and robust approach to the SO problem, particularly in large-scale networks.

4.3 Detail of SSDO

The *SD Selection* component plays a critical role in determining the sequence of SDs for the *Split Ratio Modification* component. A naive approach is to traverse all SDs in a fixed order. However, this is inefficient because many SDs have no impact on MLU, meaning their split ratios can be adjusted without affecting the optimization goal. As a result, computational resources are wasted on updating them.

To address this inefficiency, we leverage the mathematical relationship between MLU and SDs. As shown in Equation (10), the utilization rate of a link $i \to j$ is influenced by up to

Algorithm 1: Balanced Binary Search Method (BBSM)

Input: c, f_{skd}, s, d, D.
Output: Updated split ratio f_{skd}.

Initialize Q, u^{ub}, $\underline{u} \leftarrow 0$, $\bar{u} \leftarrow u^{ub}$, continue $\leftarrow$ **TRUE**;
while *continue* **do**
  $u \leftarrow \frac{\bar{u}+\underline{u}}{2}$;
  Calculate $\bar{f}^b_{skd}(u)$
  if $\sum_{k\in V} \bar{f}^b_{skd}(u) \geq 1$ **then**
    $\bar{u} \leftarrow u$;
  end
  else
    $\underline{u} \leftarrow u$;
  end
  if $|\bar{u} - \underline{u}| < \varepsilon$ **then**
    continue $\leftarrow$ **FALSE**;
  end
end
Set $u \leftarrow \bar{u}$;
Set $f_{skd} \leftarrow \bar{f}^b_{skd}(\bar{u})$;
return f_{skd};

Algorithm 2: Sequential Source-Destination Optimization (SSDO)

Input: c, D.
Output: Optimized split ratios.

Initialize split ratios and calculate the utilization;
Set continue $\leftarrow$ **TRUE**;
while *continue* **do**
  Obtain the sequence of SDs using *SD Selection* component;
  for *each SD in the obtained sequence* **do**
    Call the *Split Ratio Modification* component to update the split ratio;
    Update utilization;
  end
  if $opt - \max_{i,j\in V} u_{ij} \leq \varepsilon_0$ **then**
    continue $\leftarrow$ **FALSE**;
  end
  else
    Update opt: opt $\leftarrow \max_{i,j\in V} u_{ij}$;
  end
end
return *Optimized split ratios.*

$2|V|-3$ SDs. This implies that focusing on the SDs associated with the edges exhibiting the highest MLU can effectively reduce the MLU without needing to process all SDs. If any specific SD is restricted from using this link, it can simply be excluded from the calculation.

$$u_{ij} = \frac{\sum_{k\in V} f_{ijk} \cdot D_{ik} + \sum_{k\in V} f_{kij} \cdot D_{kj}}{c_{ij}}. \tag{10}$$

Based on this insight, the collaborative workflow of the *SD Selection* and *Split Ratio Modification* components is designed to prioritize efficiency.

1. ***SD Selection* component.** The *SD Selection* component identifies the edges demonstrating the highest utilization. Subsequently, it calculates the SDs associated with these edges and organizes them into a processing queue using a specified prioritization rule (e.g., frequency of occurrence).

2. ***Split Ratio Modification* component.** The *Split Ratio Modification* component processes the SDs in the queue one by one, adjusting their split ratios using BBSM to reduce MLU.

3. **Termination check.** After processing all SDs in queue, SSDO evaluates whether the MLU has decreased. If the amount of MLU reduction is less than ε_0, the algorithm terminates. Otherwise, the *SD Selection* component recalculates the SD queue.

The detailed steps of SSDO are summarized in Algorithm 2, which illustrates the interaction between two components. This collaborative design ensures that computational resources are focused on the most critical SDs, thereby improving the overall efficiency of the algorithm.

4.4 SSDO Deployment Strategies

Initialization modes. SSDO supports two initialization modes: hot-start and cold-start. In hot-start mode, SSDO uses TE configurations generated by other algorithms as the initial split ratios. The MLU in SSDO does not increase during the optimization process, guaranteeing that the solution quality is at least as good as the initial configuration. In the cold-start mode, SSDO initializes configurations according to predefined rules. Among various methods tested, directing all demands along the shortest path is identified as the most effective strategy due to its flexibility for subsequent optimization. Unless otherwise stated, all experiments in this paper adopt this cold-start method. For real-world deployment, a hybrid approach can be adopted: both hot-start and cold-start SSDO can be executed in parallel, and the system selects the best solution when the time limit is reached.

Early termination. SSDO achieves rapid MLU improvements during the early stages of optimization, making early termination a practical strategy, particularly in time-sensitive scenarios. This is especially effective in hot-start mode, particularly when initialized with DL-based solutions, which quickly generate feasible configurations for SSDO to refine with minimal computation. For deployment, an adaptive early termination mechanism can be implemented based on a predefined time threshold. This ensures that SSDO balances computation time and optimization quality efficiently.

Path-based formulation. For multi-hop scenarios (i.e., paths with three or more hops), SSDO must be extended to the path-based formulation, as detailed in Appendix B. This formulation introduces incidence matrices to map split ratios to SDs, paths, and edges, enabling the model to capture multi-hop routing behaviors accurately. When the topology involves only one- or two-hop connections, adopting the path-based formulation is optional and can be decided based on the number of available candidate paths: if only a few exist, the path-based form can substantially reduce the problem size; otherwise, the original SSDO formulation remains preferable for its superior computational efficiency.

5 Evaluation

In this section, we present a comprehensive evaluation of SSDO. First, we outline the methodology and test system used in our experiments in §5.1. Next, we compare SSDO against other TE approaches, focusing on both TE quality and computational efficiency in §5.2. Following this, §5.3 and §5.4 evaluate SSDO's effectiveness in managing link failures and adapting to dynamic traffic changes, respectively. Additionally, we assess the performance of SSDO on WAN in §5.5. The experiment about hot-start mode and early termination are detailed in §5.6. Finally, in §5.7, we analyze the necessity of SSDO's individual components through ablation studies.

5.1 Methodology

Topologies. Our evaluation covers two types of topologies: Meta's DCN [39], including Top-of-Rack (ToR) and Point of Delivery (PoD) levels, and two WAN topologies, UsCarrier and Kdl, from the Internet Topology Zoo [24]. Shortest paths between SD pairs are precomputed using Yen's algorithm [1]. Meta's DCN topologies are modeled as complete graphs K_n of sizes 4, 8, 155, and 367, corresponding to DB and WEB clusters under centralized TE. WAN topologies serve only as auxiliary demonstrations. Table 1 summarizes the nodes, edges, and paths. For ToR-level DCNs, we test both per-pair 4-path limits and all-path settings.

Traffic data. In the study of Meta topologies, we utilize the publicly available one-day traffic trace provided by [39]. For the PoD-level topology, traffic traces are aggregated into 1-second snapshots of the inter-PoD traffic matrix, whereas for the ToR-level topology, aggregation is performed over 100-second intervals to generate the inter-ToR traffic matrix. For the UsCarrier and Kdl topologies from Topology Zoo, where no public traffic traces are available, we employ a gravity model [7, 38] to generate synthetic traffic.

Baselines. We select the following baselines to evaluate SSDO, with parameters chosen based on comprehensive considerations: (1) **LP-all:** Commercial LP solvers (Gurobi [18]) directly solve TE, providing a theoretically optimal MLU. (2) **LP-top** [32]: This method focuses on the top α% demands while routing the rest via shortest paths. Based on a trade-off between computational efficiency and solution quality, we select $\alpha = 20$ for all subsequent tests. (3) **POP** [33]: This method decomposes the optimization problem into k subproblems, with each subproblem handling $1/k$ of the total demands while the capacity of each link is scaled down to $1/k$ of its original value. After balancing computational cost and performance, we set $k = 5$ for the evaluations. (4) **DOTE-m (DOTE [35], Figret [30]):** These methods take the traffic matrix as input and directly output the split ratios using a fully connected neural network. The models are trained with MLU as the loss function, optimizing traffic allocation to minimize congestion. In our experiments, we modify DOTE to take the current traffic matrix as input, referring to it as DOTE-m. (5) **Teal** [44]: A reinforcement learning-based method using a shared policy network to allocate each SD's demands independently. The shared network significantly reduces the problem scale, making it suitable for large-scale networks. The effectiveness of SSDO's individual modules is assessed separately via ablation baselines in Section 5.7.

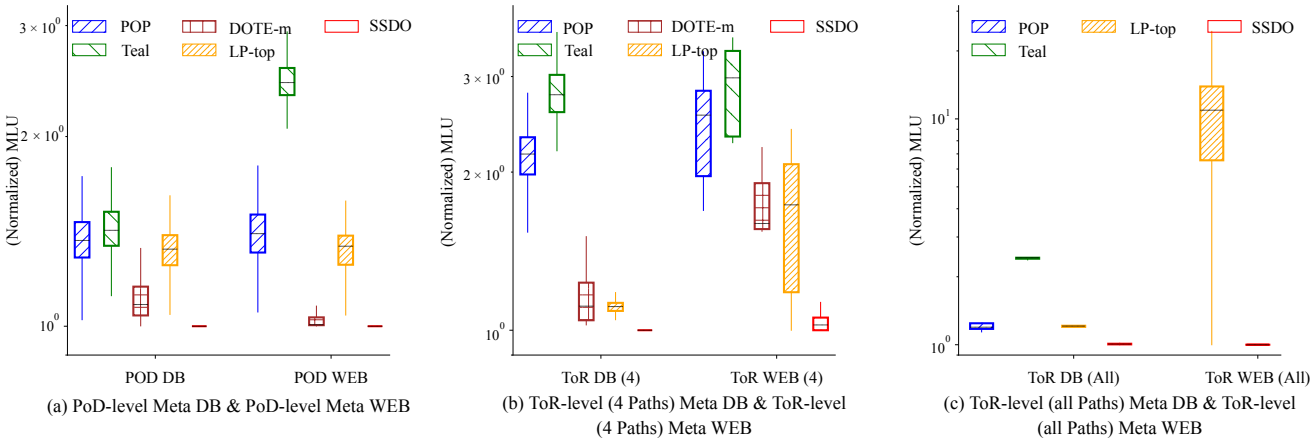
	#Type	#Nodes	#Edges	#Paths
Meta DB	PoD-level DC	4	12	3
	ToR-level DC	155	23870	4
	ToR-level DC	155	23870	154
Meta WEB	PoD-level DC	8	56	7
	ToR-level DC	367	134322	4
	ToR-level DC	367	134322	366
UsCarrier	WAN	158	378	4
Kdl	WAN	754	1790	2

Table 1: Network topologies in our evaluation.

Infrastructure and software. Computational experiments are conducted on an Intel® Xeon® Platinum 8260 CPU with 1 TB of memory. Additionally, three NVIDIA GeForce RTX 4090 GPUs (each with 24 GB VRAM) are used for DL-based methods, including DOTE-m and Teal. These methods are implemented and evaluated using PyTorch 2.10, which is compatible with CUDA 12.1 [34]. LP-based methods are evaluated using Gurobi 9.5.1 [18], with all reported times referring to TotalTime (including both model construction and solving, which is only marginally larger than RunTime in our setting). All implementations, including SSDO, are developed in Python 3.8.

5.2 Comparison with Other TE Methods

This section evaluates the TE performance and computation time across various topologies in Figure 5 and Figure 6, focusing on normalized MLU relative to the LP-all method and computational time for each scheme. Both figures are pre-



(a) PoD-level Meta DB & PoD-level Meta WEB

(b) ToR-level (4 Paths) Meta DB & ToR-level (4 Paths) Meta WEB

(c) ToR-level (all Paths) Meta DB & ToR-level (all Paths) Meta WEB

Figure 5: TE quality performance of SSDO and other baseline. Methods order: POP, Teal, DOTE-m, LP-top, SSDO. In ToR-level (all paths) DB: DOTE-m failed. In ToR-level (all paths) WEB: DOTE-m, Teal and POP failed.

sented on logarithmic scales for clarity. Notably, in the ToR-level Meta WEB topology (all paths), where LP-all fails to yield a feasible solution within the set time limitation (45,000 seconds), SSDO's MLU serves as the normalization baseline. The results demonstrate SSDO's exceptional balance between solution quality and efficiency, particularly in large-scale topologies. Key findings include:

LP-all: Designed to provide optimal MLU solutions, LP-all serves as a benchmark for TE quality. However, its computation time increases exponentially with problem scale, becoming impractical even in medium-sized topologies. For instance, LP-all requires nearly 200s for the ToR-level Meta WEB (4 paths) topology and nearly 1,000s for the ToR-level Meta DB (all paths). In the ToR-level Meta WEB topology (all paths), LP-all fails to yield a feasible solution within time limitation, and thus its results are omitted from that topology.

POP: POP demonstrates unsatisfying TE performance due to its decomposition strategy, which isolates subproblems without accounting for coupling. While this approach can be effective for maximizing network flow, it is unsuitable for minimizing MLU. In the ToR-level WEB (4 paths) topology, POP's MLU is 2.44× higher than SSDO's. Furthermore, in the ToR-level Meta WEB topology (all paths), its solving time exceeds time limitations, making it infeasible for large-scale networks. Consequently, POP's results are not included in Figure 5 or Figure 6 for this topology.

LP-top: LP-top improves upon LP-all by prioritizing the top $\alpha\%$ of demands, enabling better routing decisions for high-priority traffic. However, its simplistic handling of low-priority demands leads to unsatisfying configurations, especially in complex topologies like the ToR-level WEB (all paths), where its MLU is 10.93× higher than SSDO's. Additionally, LP-top's computation time escalates with topology size, becoming impractical in large-scale scenarios.

Teal: While Teal achieves competitive efficiency in part of topologies, its TE quality remains unsatisfactory due to its design. Its shared policy structure struggles to capture the intricate demand couplings characteristic of DCNs. Moreover, Teal fails to provide feasible solutions in large-scale settings, where Video Random Access Memory (VRAM) limitations render it infeasible.

DOTE-m: DOTE-m quickly generates configurations in medium-scale topologies like ToR-level (4 paths). While its performance is inferior to SSDO, its fast inference speed provides an advantage. However, in large topologies, its fully connected network struggles with increased output dimensions and high VRAM consumption, limiting its scalability.

SSDO: SSDO achieves high-quality configurations across all tested topologies with competitive efficiency. For PoD level, despite its Python implementation, it reduces error rates below 1% within 0.3s. For ToR-level WEB (4 paths), SSDO outperforms alternatives by reducing errors by 57% in around 2s. In the challenging all-path Meta WEB topology, where most methods fail, SSDO completes optimization in 165s with robust accuracy. All tests use cold-start mode (§4.4). Notably, SSDO supports early termination, enabling high-quality solutions under time constraints (§5.6). Further improvements in implementation could enhance its performance.

5.3 Coping with Network Failures

Figure 7 compares the performance of SSDO and other TE methods under different levels of random link failures in the ToR-level WEB topology (4 paths). Naturally, LP-all remains theoretically optimal even with a small number of failures, while maintaining stable MLU. Other LP-based methods exhibit poor performance, failing to meet practical requirements.

In addition, DOTE-m experiences a noticeable increase in MLU as failures grow. This degradation arises because its training data is derived from failure-free networks, rather than

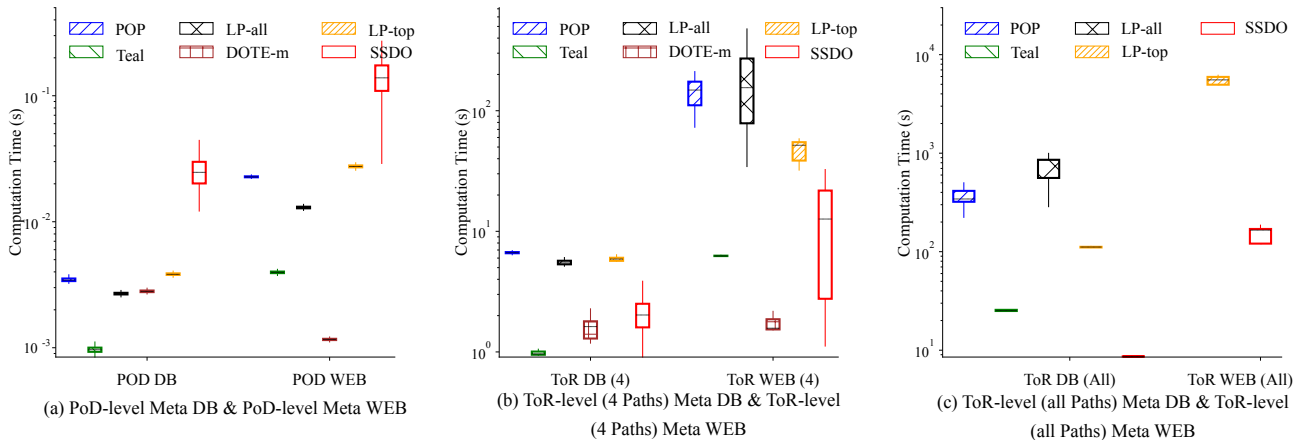


Figure 6: Computation time performance. Methods order: POP, Teal, LP-all, DOTE-m, LP-top, SSDO. In ToR-level (all paths) DB: DOTE-m failed. In ToR-level (all paths) WEB: DOTE-m, Teal,LP-all and POP failed.

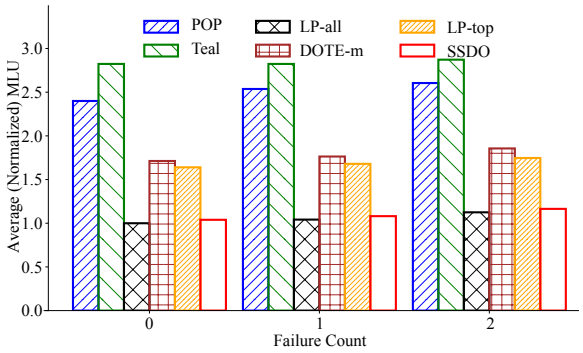


Figure 7: Coping with different numbers of random link failures on ToR-level WEB (4 paths). The y-axis represents normalized MLU using origin topology.

being intrinsic to DL-based approaches. When link failures occur, the mapping between traffic matrices and TE configurations shifts, leading to degraded performance. Indeed, recent work (e.g., Harp [6]) has shown that carefully designed DL-based models can generalize to unseen topologies and even handle failures beyond the training distribution. More broadly, besides their inference speed, a key motivation for DL-based schemes is their potential to cope with prediction errors and uncertainty in traffic demands.

By contrast, SSDO achieves performance close to LP-all while maintaining strong adaptability and resilience to link failures. Unlike DL-based methods, SSDO does not require training data, making it a robust and practical choice for handling failures in dynamic network environments.

5.4 Robustness to Demand Changes

To assess the impact of temporal fluctuations on TE methods, we introduce different levels of variation into the traffic matrix. For each demand, we calculate the variance of its changes across consecutive time slots and scale it by factors of 2, 5, and 20. Using these scaled variances, we define zero-mean normal distributions, from which random samples are drawn and added to each demand in every time interval.

As shown in Figure 8, SSDO maintains stable and high-quality performance across all fluctuation levels, demonstrating its robustness to temporal variations. LP-top and POP exhibit relatively stable performance, indicating that their optimization strategies are less sensitive to fluctuations. However, POP shows irregular variations, which stem from its algorithmic design. Interestingly, LP-top's performance slightly improves as fluctuations increase, likely because larger variations amplify the proportion of high-demand traffic, enabling LP-top to allocate resources more efficiently.

In contrast, DOTE-m and Teal experience a clear decline in performance as fluctuation levels increase. This degradation is likely caused by the growing discrepancy between the perturbed traffic matrices and the historical ones used for training, limiting generalization to unseen traffic patterns.

5.5 SSDO for WAN

To demonstrate the generality of SSDO beyond DCNs, we further evaluate its path-based formulation (Appendix B) on two WAN topologies. Figure 9 presents the results, comparing SSDO with various TE methods in terms of computation time and solution quality. SSDO consistently achieves high-quality solutions with competitive solving times, demonstrating its ability to generalize effectively to WAN settings.

In UsCarrier, SSDO achieves lower MLU than LP-based methods (POP, LP-top) while maintaining a solving time under one second, comparable to DL-based methods (DOTE-m, Teal). This efficiency highlights SSDO's practicality in small-scale WANs. In KDL, SSDO reduces MLU by 9% compared

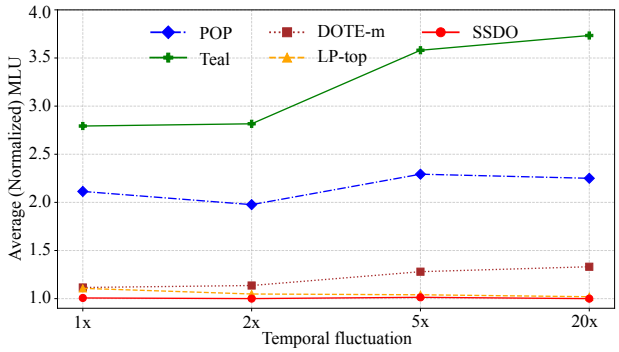


Figure 8: Coping with Temporal fluctuation on Meta ToR-level DB (4 paths). The y-axis represents the MLU normalized by that of the LP-all using perturbed traffic matrix.

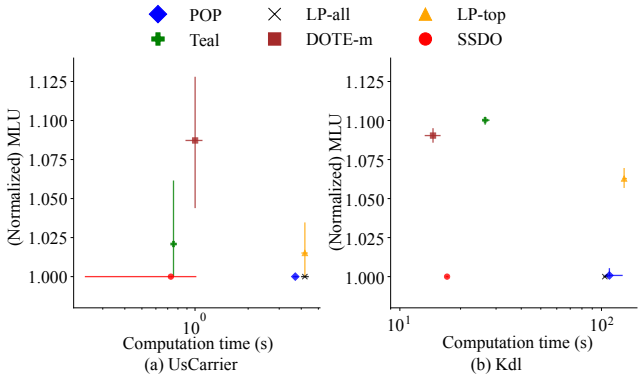


Figure 9: Performance of SSDO and baselines in WAN. The y-axis represents the normalized MLU. The x-axis represents the computation time (in seconds) on a logarithmic scale.

to DOTE-m and Teal while slightly outperforming POP. Although its solving time is marginally longer than DOTE-m, it remains significantly faster than LP-based methods. Notably, Teal's solving time is higher than reported in prior work [44], likely due to cases where it outputs all-zero split ratios, requiring additional corrections.

5.6 Hot-start Initialization and Early Termination in SSDO

Figure 10 shows the evolution of the MLU error relative to the optimal MLU throughout the SSDO optimization process. The y-axis represents the normalized error reduction, and the x-axis represents the normalized optimization time, ranging from 0 (start) to 1 (completion). The results demonstrate that SSDO achieves rapid error reductions during the initial stages of optimization across all topologies. This characteristic provides strong support for the practicality of hot-start mode and early termination strategies, enabling high-quality solutions to be obtained with constrained computation time.

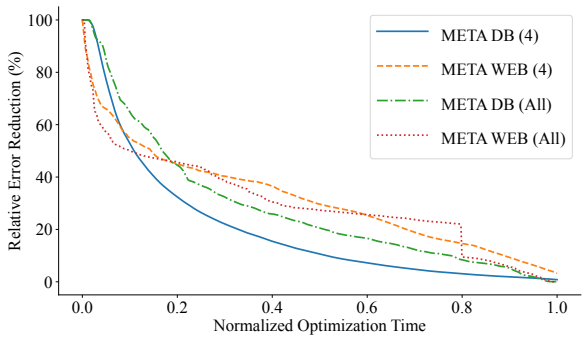


Figure 10: Relative error reduction of MLU in topologies.

The effectiveness of hot-start SSDO is further validated in Appendix E, which compares hot-start and cold-start modes. As shown in Figures 11-12, hot-start SSDO—initialized with DOTE-m solutions—outperforms DOTE-m and approaches the performance of cold-start SSDO, while requiring relatively less computation time. However, in some cases, cold-start SSDO completes optimization faster than hot-start mode due to the overhead of generating the initial solution by DOTE-m. This suggests that in practical deployment, running both hot-start and cold-start SSDO in parallel and selecting the better-performing solution can further enhance efficiency. Additionally, Table 4 demonstrates that even with early termination, hot-start SSDO—leveraging DOTE-m's solutions—reduces MLU by up to 35.9% within just 3 seconds, confirming SSDO's adaptability to different time constraints.

5.7 Ablation Study of SSDO

We perform an ablation study to assess the impact of SSDO's key features on its overall performance.

Design of BBSM. The BBSM accelerates the SO process and identifies globally beneficial solutions. In the SSDO/LP variant, subproblems are solved using LP solver (Gurobi), but split ratios are refined by BBSM to maintain consistency. Table 2 shows that SSDO/LP is significantly slower than SSDO, demonstrating the efficiency of BBSM. Meanwhile, SSDO/LP-m employs split ratios calculated by Gurobi directly. As shown in Table 3, these ratios lead to a higher MLU, emphasizing the necessity of using balanced solutions.

Design of SD Selection. SSDO optimizes SDs associated with edges of the highest real-time utilization, focusing on bottlenecks in each iteration. By contrast, SSDO/Static traverses all SDs per iteration. Table 2 shows that SSDO/Static variant incurs substantially longer computation times, proving the efficiency of our prioritization strategy.

6 Related Work

TE in DCNs and WANs. TE is critical for optimizing network performance, ensuring fairness, and preventing link overuti-

Topology	SSDO	SSDO/LP	SSDO/Static
PoD-level DB	0.03	0.15	1.27
PoD-level WEB	0.14	1.57	3.81
ToR-level DB (4)	2.16	202.28	184.37
ToR-level WEB (4)	17.95	2796.84	3374.04

Table 2: Comparison of computation Time (seconds) Across Variants

Topology	SSDO	SSDO/LP-m
PoD-level DB	1.00	1.10
PoD-level WEB	1.00	1.44
ToR-level DB (4)	1.01	3.41
ToR-level WEB (4)	1.00	5.06

Table 3: Comparison of MLU Across Variants

lization in both DCNs and WANs. Hardware-based TE methods such as ECMP [20, 47] and WCMP [11, 50] are commonly employed to efficiently utilize bandwidth. However, these methods struggle with asymmetry and heterogeneity in traffic patterns. To overcome these challenges, SDN-based centralized TE systems [7, 43] have gained popularity by addressing global optimization objectives such as MLU. While effective, scaling these systems to large, dynamic networks remains a significant challenge.

Machine Learning in TE. Machine learning (ML) has been applied in TE primarily for two purposes: prediction of traffic demand and direct configuration of TE. The first category uses predictive models to estimate future traffic based on historical data [13, 27, 46, 49], which are then input into optimization algorithms to compute TE configurations. The second category learns a mapping from traffic to TE configurations, as demonstrated by methods like DOTE [35] and others [30, 42, 44]. Although these approaches leverage the ability of ML to model complex relationships, they face scalability challenges in large networks and struggle to handle unexpected traffic bursts, limiting their applicability in dynamic and large-scale networks.

TE Acceleration. TE acceleration have been extensively studied to address the computational challenges of large-scale TE. For SDN environments, methods such as Teal [44] and POP [33] support both maximum flow and MLU minimization objectives, while NCFlow [1] is specifically tailored for maximum flow optimization. In hybrid SDN scenarios [13, 23, 40], Agarwal et al. [2] proposed a greedy SDN switch placement approach combined with a fully polynomial-time approximation scheme to optimize traffic split ratios. Building on this, Guo et al. [15–17] introduced heuristic algorithms that jointly optimize OSPF link weights and SDN traffic splits, effectively reducing MLU in hybrid networks. Despite these advancements, achieving both efficiency and high TE quality remains a significant challenge in large-scale and dynamic network environments.

7 Discussion

Analysis of optimality. As noted in section 5, there remains a small but notable gap between the MLU achieved by SSDO and the theoretical optimum. This gap arises because SSDO may terminate when it encounters a particular situation, referred to as deadlock. In Appendix F, we provide an illustrative example of deadlock, define the phenomenon in detail, and demonstrate how it affects SSDO performance. We also discuss why deadlock rarely affects the reliability of SSDO.

Analysis of objective. SSDO is effective for MLU, since its bottleneck-driven monotonicity enables clean decomposition and efficient solution. Other objectives, such as throughput, lack these properties, though prior work like PCF [22] shows they can be related to MLU within a unified framework. Thus, SSDO's strong guarantees hold for MLU, while extensions to other metrics remain possible only with approximation.

Analysis of Implementation: In software-defined TE systems [44], as described in section G, SSDO is used within the TE controller to solve optimization problems. The TE controller takes current traffic demands and network topology as inputs, and produces traffic allocations as output. Recently, some DL-based systems have begun using historical traffic data as input. We believe SSDO could potentially be applied to these systems.

8 Conclusion

In this work, we introduce SSDO, a novel TE acceleration algorithm designed for large-scale DCNs. SSDO employs a sequential subproblem-solving strategy, where each subproblem optimizes the split ratios for a specific source-destination (SD). The subproblem order is dynamically adjusted based on real-time utilization to accelerate convergence. Each subproblem is solved using the Balanced Binary Search Method (BBSM), which efficiently identifies the most balanced and MLU-minimizing solution. To further improve efficiency, SSDO supports hot-start initialization, leveraging existing TE solutions as starting points, and early termination, ensuring high-quality solutions within limited computation time. Experimental results demonstrate that SSDO significantly outperforms existing methods, achieving superior TE quality while maintaining competitive computation efficiency. These features make SSDO a scalable and robust solution for large-scale TE in real-world networks.

9 ACKNOWLEDGMENTS

We thank our anonymous reviewers for their insightful comments and Ryan Beckett for shepherding this work. We also thank Zhangheng Liu, Haozhe Li, Fanfan Li, Yuqian Ying, for their helpful feedback. This work was supported by "the Fundamental Research Funds for the Central Universities".

References

[1] Firas Abuzaid, Srikanth Kandula, Behnaz Arzani, Ishai Menache, Matei Zaharia, and Peter Bailis. Contracting wide-area network topologies to solve flow problems quickly. In *18th USENIX symposium on networked systems design and implementation (NSDI 21)*, pages 175–200. USENIX Association, April 2021.

[2] Sugam Agarwal, Murali Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *2013 Proceedings IEEE INFOCOM*, pages 2211–2219, 2013.

[3] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30, 2014.

[4] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 19, USA, April 2010. USENIX Association.

[5] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. CONGA: distributed congestion-aware load balancing for datacenters. *SIGCOMM Comput. Commun. Rev.*, 44(4):503–514, 2014.

[6] Abd AlRhman AlQiam, Yuanjun Yao, Zhaodong Wang, Satyajeet Singh Ahuja, Ying Zhang, Sanjay G. Rao, Bruno Ribeiro, and Mohit Tawarmalani. Transferable Neural WAN TE for Changing Topologies. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, pages 86–102, New York, NY, USA, August 2024. Association for Computing Machinery.

[7] David Applegate and Edith Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 313–324, New York, NY, USA, 2003. Association for Computing Machinery.

[8] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. Optimal oblivious routing in polynomial time. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 383–388, New York, NY, USA, 2003. Association for Computing Machinery.

[9] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. MicroTE: fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, pages 1–12, Tokyo Japan, December 2011. ACM.

[10] Jeremy Bogle, Nikhil Bhatia, Manya Ghobadi, Ishai Menache, Nikolaj Bjørner, Asaf Valadarsky, and Michael Schapira. Teavar: striking the right utilization-availability balance in wan traffic engineering. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 29–43, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Yingying Cheng and Xiaohua Jia. Namp: Network-aware multipathing in software-defined data center networks. *IEEE/ACM Transactions On Networking*, 28(2):846–859, 2020.

[12] Marco Chiesa, Gábor Rétvári, and Michael Schapira. Lying Your Way to Better Traffic Engineering. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 391–398, New York, NY, USA, 2016. Association for Computing Machinery.

[13] Kaihui Gao, Dan Li, Li Chen, Jinkun Geng, Fei Gui, Yang Cheng, and Yue Gu. Incorporating intra-flow dependencies and inter-flow correlations for traffic matrix prediction. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2020.

[14] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM Computer Communication Review*, 38(4):75–86, October 2008.

[15] Yingya Guo, Huan Luo, Zhiliang Wang, Xia Yin, and Jianping Wu. Routing optimization with path cardinality constraints in a hybrid sdn. *Computer Communications*, 165:112–121, 2021.

[16] Yingya Guo, Weipeng Wang, Han Zhang, Wenzhong Guo, Zhiliang Wang, Ying Tian, Xia Yin, and Jianping Wu. Traffic engineering in hybrid software defined network via reinforcement learning. *Journal of Network and Computer Applications*, 189:103116, 2021.

[17] Yingya Guo, Zhiliang Wang, Xia Yin, Xingang Shi, and Jianping Wu. Traffic engineering in sdn/ospf hybrid network. In *2014 IEEE 22nd international conference on network protocols*, pages 563–568. IEEE, 2014.

[18] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2023.

[19] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pages 15–26, 2013.

[20] Christian Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. Request for Comments RFC 2992, Internet Engineering Task Force, November 2000.

[21] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[22] Chuan Jiang, Sanjay Rao, and Mohit Tawarmalani. PCF: Provably Resilient Flexible Routing. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, pages 139–153, New York, NY, USA, July 2020. Association for Computing Machinery.

[23] Sajad Khorsandroo, Adrián Gallego Sánchez, Ali Saman Tosun, José M Arco, and Roberto Doriguzzi-Corin. Hybrid sdn evolution: A comprehensive survey of the state-of-the-art. *Computer Networks*, 192:107981, 2021.

[24] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[25] Mario Koppen. The curse of dimensionality. In *Proceedings of Online World Conference on Soft Computing in Industrial Applications (WSC)*, volume 1, pages 4–8, 2000.

[26] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 1–14, New York, NY, USA, 2015. Association for Computing Machinery.

[27] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-Oblivious traffic engineering: The road not taken. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 157–170, Renton, WA, April 2018. USENIX Association.

[28] Yin Tat Lee and Aaron Sidford. Efficient Inverse Maintenance and Faster Algorithms for Linear Programming. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 230–249, Berkeley, CA, USA, October 2015. IEEE.

[29] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. Traffic engineering with forward fault correction. *SIGCOMM Comput. Commun. Rev.*, 44(4):527–538, August 2014.

[30] Ximeng Liu, Shizhen Zhao, Yong Cui, and Xinbing Wang. FIGRET: Fine-grained robustness-enhanced traffic engineering. In *Proceedings of the ACM SIGCOMM 2024 Conference*, Acm Sigcomm '24, pages 117–135, Sydney, NSW, Australia and New York, NY, USA, 2024. Association for Computing Machinery.

[31] D. Mitra and K.G. Ramakrishnan. A case study of multiservice, multipriority traffic engineering design for data networks. In *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99. (Cat. No.99CH37042)*, volume 1b, pages 1077–1083, Rio de Janeireo, Brazil, 1999. IEEE.

[32] Pooria Namyar, Behnaz Arzani, Ryan Beckett, Santiago Segarra, Himanshu Raj, and Srikanth Kandula. Minding the gap between fast heuristics and their optimal counterparts. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 138–144, Austin Texas, November 2022. ACM.

[33] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. Solving Large-Scale Granular Resource Allocation Problems Efficiently with POP. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 521–537, Virtual Event Germany, October 2021. ACM.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy,

Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, page Article 721. Curran Associates Inc., 2019.

[35] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. DOTE: Rethinking (predictive) WAN traffic engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1557–1581, Boston, MA, April 2023. USENIX Association.

[36] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. Jupiter evolving. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 66–85, Amsterdam Netherlands, August 2022. ACM.

[37] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. Alibaba HPN: A Data Center Network for Large Language Model Training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 691–706, Sydney NSW Australia, August 2024. ACM.

[38] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. Experience in measuring internet backbone traffic variability: Models metrics, measurements and meaning. In J. Charzinski, R. Lehnert, and P. Tran-Gia, editors, *Teletraffic Science and Engineering*, volume 5 of *Providing Quality of Service in Heterogeneous Environments*, pages 379–388. Elsevier, January 2003.

[39] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, London United Kingdom, August 2015. ACM.

[40] Rui Silva, David Santos, Flavio Meneses, Daniel Corujo, and Rui L Aguiar. A hybrid sdn solution for mobile networks. *Computer Networks*, 190:107958, 2021.

[41] Github repository containing our code. https://github.com/Yingming-Mao/SSDO, 2025.

[42] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to Route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets '17, pages 185–191, New York, NY, USA, 2017. Association for Computing Machinery.

[43] Hao Wang, Haiyong Xie, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert Greenberg. Cope: traffic engineering in dynamic networks. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, page 99–110, New York, NY, USA, 2006. Association for Computing Machinery.

[44] Zhiying Xu, Francis Y. Yan, Rachee Singh, Justin T. Chiu, Alexander M. Rush, and Minlan Yu. Teal: Learning-Accelerated Optimization of WAN Traffic Engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 378–393, New York NY USA, September 2023. ACM.

[45] Nicu Florin Zaicu, Matthew Luckie, Richard Nelson, and Marinho Barcellos. Helix: Traffic engineering for multi-controller sdn. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, SOSR '21, page 80–87, New York, NY, USA, 2021. Association for Computing Machinery.

[46] Chun Zhang, Yong Liu, Weibo Gong, Jim Kurose, Robert Moll, and Don Towsley. On optimal routing with multiple traffic matrices. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 1, pages 607–618. IEEE, 2005.

[47] Hailong Zhang, Xiao Guo, Jinyao Yan, Bo Liu, and Qianjun Shuai. Sdn-based ecmp algorithm for data center networks. In *2014 IEEE Computers, Communications and IT Applications Conference*, pages 13–18, 2014.

[48] Mingyang Zhang, Jianan Zhang, Rui Wang, Ramesh Govindan, Jeffrey C. Mogul, and Amin Vahdat. Gemini: Practical Reconfigurable Datacenter Networks with Topology and Traffic Engineering, October 2021.

[49] Yuntian Zhang, Ning Han, Tengteng Zhu, Junjie Zhang, Minghao Ye, Songshi Dou, and Zehua Guo. Prophet: Traffic engineering-centric traffic matrix prediction. *IEEE/ACM Transactions on Networking*, 32(1):822–832, 2024.

[50] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. WCMP: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, pages 1–14, Amsterdam The Netherlands, April 2014. ACM.

Appendix

A Traffic Engineering in Path Form

The traffic engineering (TE) problem in path form represents the flow distribution across candidate paths for each source-destination (SD). This formulation reduces the number of decision variables in constrained scenarios but requires additional structures to map paths, edges, and nodes.

A.1 Notations

- $G = (V, E, c)$: The network topology, where:
  - V: The set of nodes.
  - E: The set of edges.
  - c_e: The capacity of link $e \in E$.
- D_{sd}: The traffic demand from source s to destination d, expressed as a scalar value.
- P_{sd}: The set of candidate paths between source s and destination d. Each path $p \in P_{sd}$ consists of a sequence of links.
- f_p: The split ratio for path $p \in P_{sd}$, representing the fraction of D_{sd} allocated to path p. It satisfies $\sum_{p \in P_{sd}} f_p = 1$.

A.2 Optimization Model

The goal of TE is to minimize the maximum link utilization (MLU), ensuring balanced traffic distribution and avoiding congestion. The problem is formulated as follows:

$$\min_{f_p} \quad \max_{e \in E} \frac{\sum_{s,d \in V} \sum_{p \in P_{sd}, e \in p} D_{sd} \cdot f_p}{c_e}, \tag{11}$$

$$\text{s.t.} \quad \sum_{p \in P_{sd}} f_p = 1, \quad \forall s, d \in V, \tag{12}$$

$$0 \leq f_p \leq 1, \quad \forall s, d \in V, \forall p \in P_{sd}. \tag{13}$$

Equation (11) minimizes the MLU across all network links. Equation (12) ensures that the total split ratios sum to one for each SD, while Equation (13) enforces non-negativity and normalization constraints on the split ratios.

B SSDO in Path Form

The Sequential Source-Destination Optimization (SSDO) minimizes the MLU u by iteratively adjusting path split ratios f_p. The process consists of the following steps:

1. **Initialization**:
   - Set initial split ratios f_p for all paths, ensuring:
   $$\sum_{p \in P_{sd}} f_p = 1, \quad \forall s, d \in V.$$
   - Compute the initial link utilization by
   $$U[e] = \sum_{s,d \in V} \sum_{p \in P_{sd}, e \in p} \frac{D_{sd} f_p}{c_e}.$$
   - Set initial u_{prev}:
   $$u_{\text{prev}} = \max_{e \in E} U[e].$$
2. **Identify Congested Edges**:
   - Identify the set of edges $E_{\max} \subseteq E$ with utilization equal to the maximum u:
   $$E_{\max} = \{e \in E \mid U[e] = u_{prev}\}.$$
3. **Map to SD**:
   - For each edge $e \in E_{\max}$, identify the set of SD (s,d) whose paths P_{sd} traverse e.
4. **Update Split Ratios Using PB-BBSM**:
   - For each identified SD (s,d), apply the Path-Based Balanced Binary Search Method (PB-BBSM) to update the split ratios f_p for paths $p \in P_{sd}$.
   - The detailed steps are provided in Section C.
5. **Recompute Link Utilization**:
   - Recalculate $U[e]$ for all $e \in E$ using the updated f_p.
   - Update the maximum link utilization u:
   $$u = \max_{e \in E} U[e].$$
6. **Convergence Check**:
   - If the reduction in u satisfies:
   $$|u_{\text{prev}} - u| \leq \varepsilon_0,$$
   terminate the algorithm and return the optimized split ratios f_p and the minimized u.
   - Otherwise, set $u_{\text{prev}} = u$, return to Step 2, and continue the iterations.

C Path-Based Balanced Binary Search Method

PB-BBSM adjusts the split ratios f_p for a given SD (s,d) to minimize u, while ensuring traffic conservation. The algorithm is shown in Algorithm 3.

Algorithm 3: Path-Based Balanced Binary Search Method (PB-BBSM)

Input: Utilization matrix U, source s, destination d, demand matrix D, candidate paths P_{sd}, tolerance ε.

Output: Optimal split ratios f_p for paths $p \in P_{sd}$.

Initialize $\underline{u} \leftarrow 0$, $\bar{u} \leftarrow \max(U)$;
Initialize split ratios for all paths in P_{sd};
for *each path* $p \in P_{sd}$ **do**
  $R[e] = U[e] - \frac{D_{sd} f_p}{c_e}, \forall e \in p$;
end
while $\bar{u} - \underline{u} > \varepsilon$ **do**
  $u_{\text{mid}} \leftarrow \frac{\underline{u}+\bar{u}}{2}$;
  for *each path* $p \in P_{sd}$ **do**
    $\bar{f}_p = \min_{e \in p} \frac{(u_{\text{mid}} - R[e]) \cdot c_e}{D_{sd}}$; $\bar{f}_p^b \leftarrow \max(\bar{f}_p, 0)$;
  end
  if $\sum_{p \in P_{sd}} \bar{f}_p^b > 1$ **then**
    $\underline{u} \leftarrow u_{\text{mid}}$;
  end
  else
    $\bar{u} \leftarrow u_{\text{mid}}$;
  end
end
for *each path* $p \in P_{ij}$ **do**
  $\bar{f}_p = \min_{e \in p} \frac{(\bar{u} - R[e]) \cdot c_e}{D_{sd}}$; $\bar{f}_p^b \leftarrow \max(\bar{f}_p, 0)$;
end
$f_p \leftarrow \frac{\bar{f}_p^b}{\sum_{p \in P_{sd}} \bar{f}_p^b}, \quad \forall p \in P_{sd}$;
return f_p;

D Monotonicity of Upper Bound of the Split Ratio

This appendix establishes the nondecreasing property of $\bar{f}_{skd}(u)$ with respect to the MLU parameter u.

Notation recap. From Equations (3) and (4) in the main text:

$$T_{skd}(u) = \begin{cases} \min\{uc_{sk} - Q_{sk},\ uc_{kd} - Q_{kd}\}, & k \in P_{sd},\ k \neq d, \\ uc_{sd} - Q_{sd}, & k = d, \end{cases}$$

and

$$\bar{f}_{skd}(u) = \frac{T_{skd}(u)}{D_{sd}},$$

where

- $u \in \mathbb{R}_{\geq 0}$ is the candidate MLU value,
- $c_e \geq 0$ denotes the link capacity of edge e,
- $Q_{ij} \geq 0$ represents the background traffic on link (i, j),
- $D_{sd} > 0$ is the demand from source s to destination d.

Theorem 1. *For all $s, d, k \in V$, the function $\bar{f}_{skd}(u)$ is nondecreasing over $u \in [0, +\infty)$.*

Proof. Consider a link (i, j). The term $g_{ij}(u) = uc_{ij} - Q_{ij}$ is affine in u with slope $c_{ij} \geq 0$. Hence $g_{ij}(u)$ is nondecreasing.

If $k = d$, then

$$T_{skd}(u) = g_{sd}(u),$$

which is nondecreasing. If $k \neq d$, then

$$T_{skd}(u) = \min\{g_{sk}(u), g_{kd}(u)\},$$

the pointwise minimum of two nondecreasing functions, which is also nondecreasing.

Since $D_{sd} > 0$, dividing by D_{sd} preserves monotonicity. Therefore, $\bar{f}_{skd}(u) = T_{skd}(u)/D_{sd}$ is nondecreasing in u. □

Remark. A finite sum of nondecreasing functions remains nondecreasing, so $\sum_{k \in V} \bar{f}_{skd}(u)$ is also nondecreasing. This property justifies the feasibility check and the binary search procedure used in the main text.

E Hot-start and Early Termination Analysis

This section evaluates the performance of hot-start SSDO and the effectiveness of early termination strategies. Experiments were conducted on the ToR-level WEB topology (4 paths) topology, comparing hot-start SSDO (SSDO-hot) with cold-start SSDO (SSDO-cold) and DOTE-m. Additionally, we analyze the effect of early termination in hot-start to highlight its practicality for time-sensitive network.

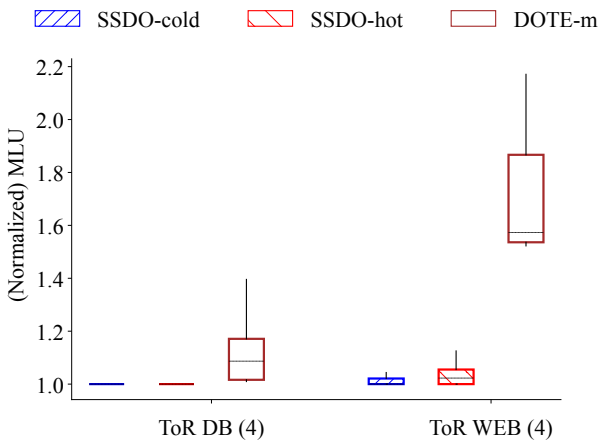


Figure 11: Comparison of SSDO-hot, SSDO-cold, and DOTE-m in MLU for ToR-level (4 paths) topologies.

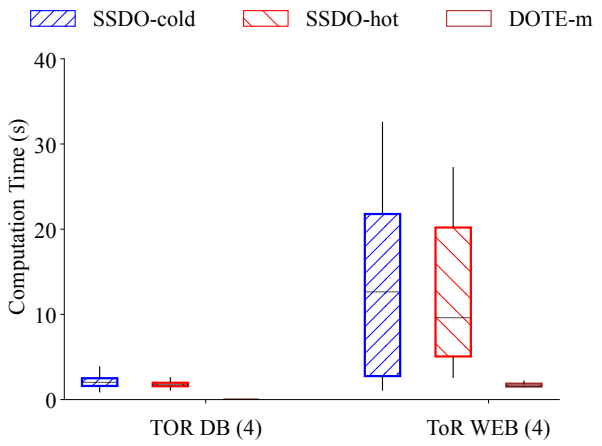


Figure 12: Comparison of SSDO-hot, SSDO-cold, and DOTE-m in computation time for ToR-level (4 paths) topologies.

E.1 Effectiveness of Hot-start Mode

In hot-start mode, SSDO initializes with solutions generated by DOTE-m, while in cold-start mode, the initial split ratios are determined based on the shortest-path strategy, as described in § 4.4. Figure 11 compares the MLU achieved by SSDO-hot, SSDO-cold, and DOTE-m. The results show that SSDO-hot consistently outperforms DOTE-m and achieves performance close to SSDO-cold. Figure 12 presents the computation time comparison. Although SSDO-hot includes the time required for DOTE-m to generate the initial solution, it runs faster than SSDO-cold in most cases. This highlights the advantage of hot-start mode in efficiently refining existing solutions while reducing computational cost.

E.2 Effectiveness of Early Termination in Hot-start Mode

To evaluate early termination, we track MLU reduction in SSDO-hot over time. Table 4 shows that SSDO-hot achieves substantial improvements within a few seconds. For instance, case 8 reduces MLU by 24.2% in 5 seconds, while cases 1 and 2 reach optimal solutions even faster. These results demonstrate that early termination in hot-start scenarios effectively balances solution quality and runtime.

E.3 Summary of Hot-Start and Early Termination Advantages

The results demonstrate SSDO's robustness in handling strict computational constraints. Hot start accelerates optimization by leveraging existing solutions, while early termination ensures high-quality results within limited time. These strategies enable SSDO to efficiently adapt to real-time performance demands, making it a practical solution for dynamic and time-sensitive environments.

Case	0s	3s	5s	10s
1	1.5637	1.0000	1.0000	1.0000
2	1.5225	1.0000	1.0000	1.0000
3	1.5384	1.1842	1.1412	1.0545
4	1.9564	1.4177	1.3047	1.1329
5	1.8368	1.6098	1.5286	1.4208
6	1.5824	1.2440	1.2035	1.0564
7	1.5291	1.2353	1.1643	1.0000
8	2.1710	1.7314	1.6415	1.4610

Table 4: Normalized MLU reduction over time in SSDO-hot for ToR-level WEB (4 paths) topology.

F Deadlock of SSDO

F.1 Definition of Deadlock in SSDO

The definition of Deadlock is as follows.

Definition 1 (Deadlock in SSDO)**.** *We call the current split-ratio configuration of SSDO to be in a* deadlock *if the following two conditions hold:*

- *For every Source-destination (SD) pair, any adjustment to its split ratios, while keeping the split ratios of all other SD demands fixed, fails to reduce the current MLU.*
- *There nevertheless exists some feasible traffic engineering configuration (e.g., by jointly adjusting split ratios for multiple SD demands) that achieves a strictly lower MLU than the current configuration.*

F.2 Illustrative Example

An anonymous reviewer provided a compelling example that highlights this phenomenon. Consider a directed ring with $n = 8$ nodes labeled A through H, with clockwise edges $\{AB, BC, CD, DE, EF, FG, GH, HA\}$, each of capacity 1. In addition, we add "skip" edges that connect every second node (e.g., $AC, BD, CE, DF, EG, FH, GA, HB$), each of infinite capacity. Each demand is placed between adjacent nodes in the clockwise direction, and each demand has two possible paths: 1) the direct one-hop edge (e.g., AB), 2) a long detour around the rest of the ring (e.g., $ACDEFGHB$). The demand for each clockwise pair of nodes is $1/5$, e.g., $A \to B, B \to C, \ldots, H \to A$. The topology, including paths and demand information, is illustrated in Figure 13.

Under the deadlock configuration, each SD routes its entire demand along the long detour (bypassing its direct one-hop edge). Let $D = 1/(n-3)$. Each detour traverses exactly $n-3$ unit-capacity ring edges, and by symmetry each ring edge lies on $n-3$ distinct detours; hence the load on every unit-capacity

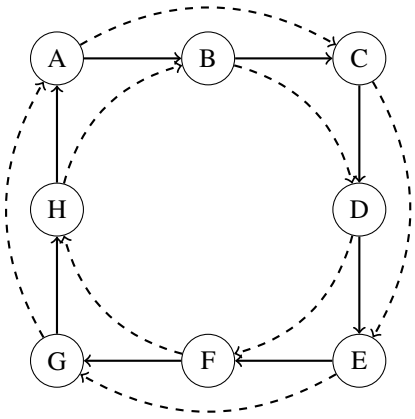


Figure 13: Directed ring topology with $n = 8$ nodes and skip edges. Each clockwise pair of nodes has a demand of $1/5$, with two paths available for each demand: the direct one-hop edge (e.g., AB) and a long detour around the ring (e.g., $ACDEFGHB$). The topology includes both solid edges (capacity 1) and dashed skip edges (infinite capacity).

ring edge equals $(n-3)D = 1$, so MLU $= 1$. Changing the split ratios of a *single* SD only removes traffic from its $n-3$ detour edges and shifts it onto its direct edge, which then exceeds capacity. As a result, the maximum utilization remains at least 1. Only a coordinated adjustment of many SDs can simultaneously reduce the load on all ring edges. By contrast, the global optimum sends each SD on its direct one-hop edge, yielding a per-edge load of $D = 1/(n-3)$ and thus MLU $= 1/(n-3)$.

This example demonstrates that SSDO may, in principle, terminate at a deadlock and miss the optimum. However, it's important to emphasize two points:

- **Pathological Initialization:** The scenario described relies on a pathological initialization where all traffic is routed along detour paths. This initialization is highly unlikely in practice. In §4.4, we introduce a cold-start initialization strategy that systematically avoids these pathological cases.
- **Proximity to Optimality:** In our numerical tests, SSDO almost always encounters deadlocks. However, even when deadlock occurs, the resulting solution is typically very close to the optimum. According to our cold-start initialization method, the initial solution will not be a deadlock (if all demands are routed through their one-hop path, but the MLU cannot be reduced by adjusting a single SD demand, the configuration is already optimal). Therefore, when deadlock occurs, it is typically far from the initial solution.

G TE Systems

Traffic engineering (TE) systems are integral to managing the flow of traffic. These systems allocate network demands to achieve high link utilization, and improve resilience to link failures. In TE systems, traffic allocation is formulated as an optimization problem, where the goal is to achieve optimal network properties such as minimum MLU, maximum throughput. The core of these systems is typically a software-defined TE controller, as illustrated in Figure 14.

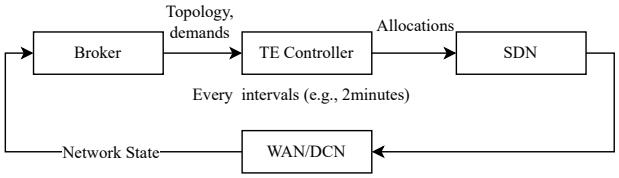


Figure 14: Control loop of traffic engineering. The TE controller periodically receives demand and topology inputs, solves the optimization problem, and updates router configurations through SDN.

The TE controller operates in a periodic control loop, as depicted in Figure 14. It receives real-time traffic demands and network topology information from the bandwidth broker at regular intervals (e.g., every two minutes). The TE controller then solves the optimization problem based on these inputs and calculates the optimal traffic allocations. Finally, the traffic allocations are translated into router configurations that are deployed via Software-Defined Networking, updating the routing policies.

This periodic feedback loop ensures that the network continuously adapts to changing traffic patterns and evolving network conditions. The ability to dynamically adjust traffic distribution is essential for achieving efficient network utilization and maintaining quality of service.